图文 45 基于MQ实现秒杀订单系统的异步化架构以及精准扣减



详情 评论

**基于MQ实现秒杀订单系统的异步化架构以及精准扣减库存的技术方案**

石杉老哥重磅力作：《互联网java工程师面试突击》（第3季）**【强烈推荐】**：


全程真题驱动，精研Java面试中**6大专题的高频考点**，从面试官的角度剖析面试

（点击下方蓝字试听）

**《互联网Java工程师面试突击》（第3季）**

**1、秒杀场景下的抢购流程分析**

上次我们讲到，明哥召开了一个团队内部的会议，先介绍了一下在秒杀场景下使用堆机器方案的弊端，接着讲了一下兄弟团队，也就是商品技术团队为了应对秒杀场景下的秒杀商品页面的高并发读取，做的大量的架构优化方案。

接着明哥就要继续讲订单技术团队为了应对秒杀的问题，需要进行哪些架构的优化。

首先从秒杀活动的场景入手来分析，假设我们每天在晚上8:30都有一个秒杀活动，都会主推一个特别好的商品进行3折限量秒杀抢购，比如一个价值6888的手机就3折出售，而且限量每天100个。

那么在这个8:30的时间点之前，实际上大量的用户（可能多达几十万甚至上百万）会集中登录到APP上，然后同时访问这个秒杀活动的商品页面，这个频繁访问商品页面的问题已经被商品技术团队解决掉了。

接着就是到8:30之后，一到时间，页面上会让一个立即抢购的按钮变成可以点击的状态，在那之前这个按钮是灰色的，不能点击。然后瞬间可能几十万甚至上百万人会同时点击这个按钮，尝试对后台发起请求去抢购这个商品。

在这个过程中，实际上大量的人要做的事情，就是跟之前正常购买商品一样的事情，比如下订单、支付、扣减库存以及后续一系列事情。所以在这个过程中，如果按照之前的策略，让所有请求都访问到订单系统以及订单数据库，那么不可避免的是导致订单系统和数据库压力过大。

如果为了每天一个秒杀活动就加10倍，20倍的机器，那么公司的成本就太高了。因此明哥带领的订单技术团队，就是对这个问题进行优化。

**2、用答题的方法避免作弊抢购以及延缓下单**

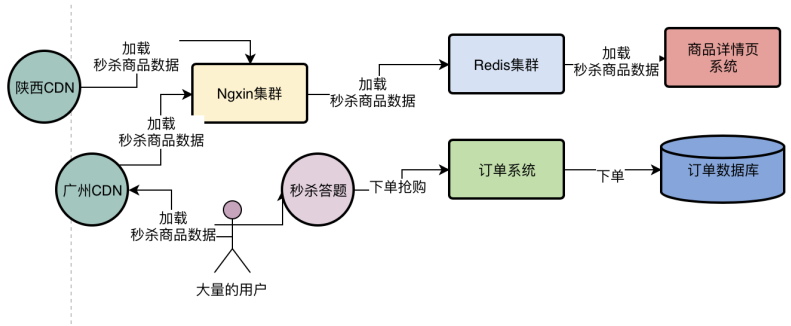
首先我们考虑第一个问题，有没有可能会有人自己写一个抢购的脚本或者作弊软件，疯狂的发送请求去抢商品

答案是肯定的，肯定是有人会写作弊的脚本或者软件。

所以一般来说，现在你要参与抢购，都会让你点击按钮之后先进行答题，就是说先弹出来一个框，让你回答一个问题，回答正确了你才能发起抢购的请求。

这个办法是非常有效的，因为首先他避免了一些作弊软件去发送抢购请求，另外就是不同的人答题的速度是不一样的，所以可以通过这个答题让不同的人发送请求的时间错开，不会在一个时间点发起请求。

所以首先就需要在客户端增加一个秒杀答题的功能，如下图所示。




**3、为秒杀独立出来一套订单系统**

接着用户的下单抢购的请求发送出去之后，会达到我们的后台系统，对于后台系统而言，我们需要思考一下，是否直接使用我们目前已有的订单系统去抗所有的请求?

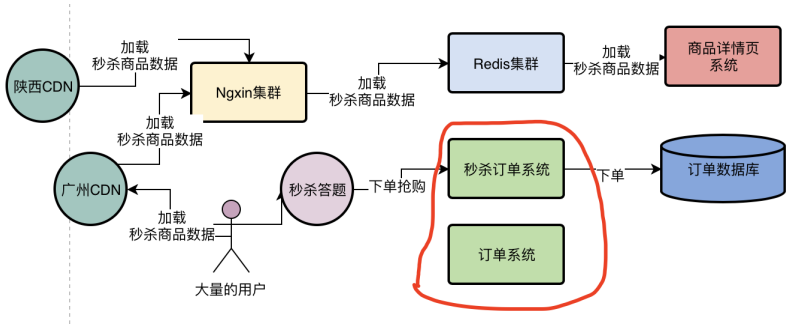
答案是否定的，这么做会有问题。

假设你有100万用户在这个时间段很活跃都会来购买商品，但是可能只有其中50万用户在参与秒杀活动，同一时间发送了大量的抢购请求到后台系统，但是同时还有很多其他的用户这个时候并不在参与秒杀系统，他们在进行其他商品的常规性浏览和下单。

因此这个时候如果你让秒杀下单请求和普通下单请求都由一套订单系统来承载，那么可能会导致秒杀下单请求耗尽了订单系统的资源，或者导致系统不稳定，然后导致其他普通下单请求也出现问题，没有办法完成的下单。

所以一般我们会对订单系统部署两个集群，一个集群是秒杀订单系统集群，一个集群是普通订单系统集群

我们看下面的图。当我们为两套系统独立部署之后，甚至可以为秒杀场景下的订单系统做很多特殊的优化。




**4、基于Redis实现下单时精准扣减库存**

然后在后台系统中我们首先需要做的一个事情，就是扣减库存。

因为大家都知道，秒杀商品一般是有数量的限制的，比如几十万人可能就抢购1万个特价商品。

所以当大量的请求到达后台系统之后，首先第一步，就可以先去扣减库存。

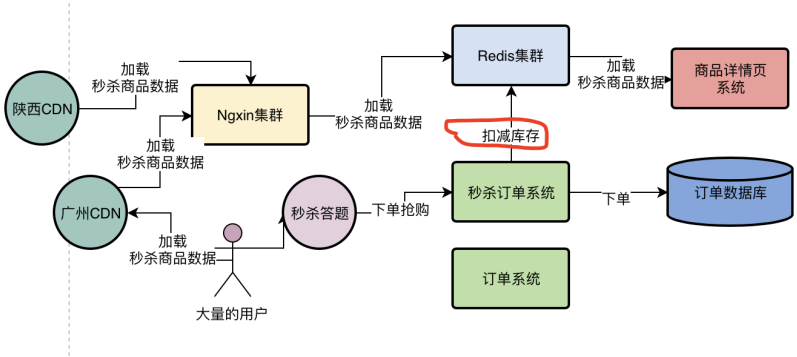
扣减库存应该怎么来扣呢？如果还是直接由订单系统调用库存系统的接口，然后访问库存数据库去扣减，那么势必导致瞬时压力过大，可能让库存系统的压力很大。

因此在秒杀场景下，一般会采用另外一个思路。

通常在秒杀场景下，**一般会将每个秒杀商品的库存提前写入Redis中**，然后当请求到来之后，就直接对Redis中的库存进行扣减

Redis是可以轻松用单机抗每秒几万高并发的，因此这里就可以抗下高并发的库存扣减

我们看下面的图：




比如我们可能总共就1万件秒杀商品，那么其实最多就是前1万个到达的请求可以成功从Redis中扣减库存，抢购到这个商品

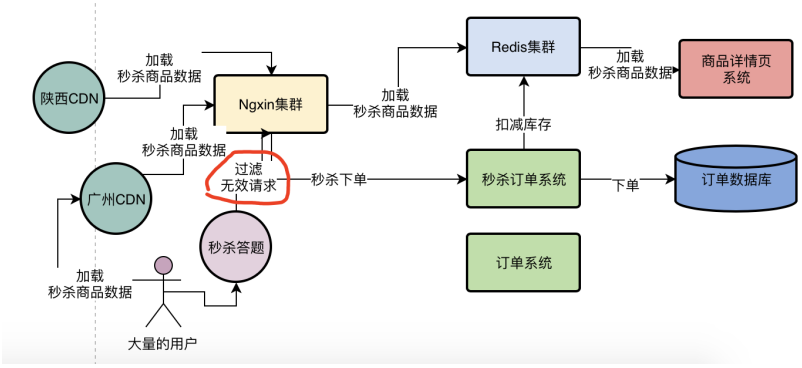
接着后续的请求在从Redis里扣减库存的时候，都会发现库存已经没了，就无法抢购到商品了。

**5、抢购完毕之后提前过滤无效请求**

其实在Redis中的库存被扣减完之后，就说明后续其他的请求都没有必要发送到秒杀系统中了，因为商品已经被抢购完毕了

此时我们可以让Nginx在接收到后续请求的时候，直接就把后续请求过滤掉。

比如一旦商品抢购完毕，可以在ZooKeeper中写入一个秒杀完毕的标志位，然后ZK会反向通知Nginx中我们自己写的Lua脚本，通过Lua脚本后续在请求过来的时候直接过滤掉，不要向后转发了。




这样的话，如果有50万人同时抢购1万件商品，其实最多就前面1万人发送的请求会抢购到商品，之后的49万请求都会在Nginx层面直接被拦截掉，过滤掉这些无效请求，返回响应告诉他们商品库存已经没了。

这样可以最大幅度削减对后端秒杀系统的请求压力。

**6、瞬时高并发下单请求进入RocketMQ进行削峰**

接着我们来考虑下，哪怕是有1万件商品同时被1万人秒杀成功了，那么可能瞬间会有1万请求涌入正常的订单系统进行后续的处理，此时可能还是会有瞬间上万请求访问到订单数据库中创建订单。

所以这个时候，完全可以引入RocketMQ进行削峰处理

也就是说，对于秒杀系统而言，如果判断发现通过Redis完成了库存扣减，此时库存还大于0，就说明秒杀成功了需要生成订单，此时就直接发送一个消息到RocketMQ中即可。

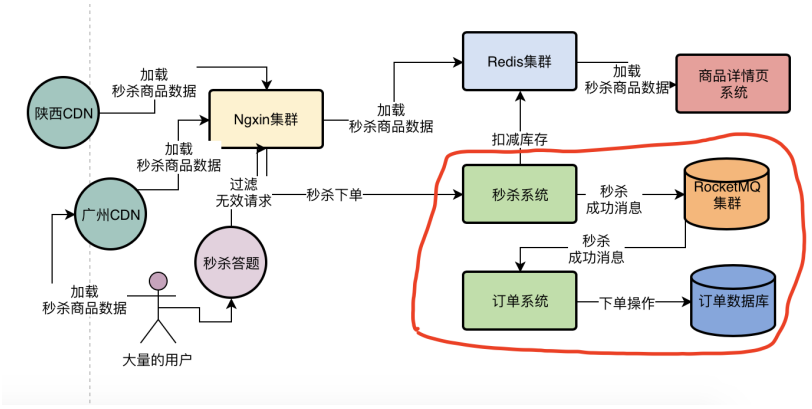
然后让普通订单系统从RocketMQ中消费秒杀成功的消息进行常规性的流程处理即可，比如创建订单，等等。

这样的话，瞬间上万并发的压力会被RocketMQ轻松抗下来，然后普通的订单系统可以根据自己的工作负载慢慢的从RocketMQ中拉取秒杀成功的消息，然后进行后续操作就可以了，不会对订单数据库造成过大的压力。

否则如果你让瞬间产生的一万或者几万的订单请求直接访问订单数据库，必然还是会让他压力过大，需要额外增加机器，那是没有必要的。

因此在这里利用RocketMQ抗下每秒几万并发的下单请求，然后让订单系统以每秒几千的速率慢慢处理就可以了，也就是延迟个可能几十秒，这些下单请求就会处理完毕。

我们看下面的图，就是这样的一个思路。




**7、秒杀架构的核心要点**

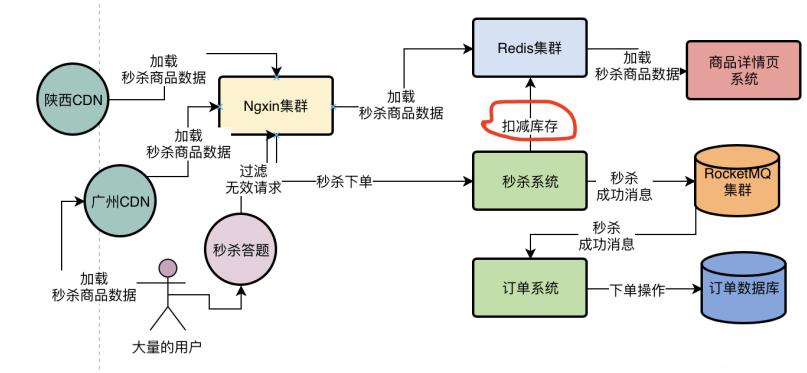
其实大家通过这篇文章的思路分析，就会清晰的看到，对于一个秒杀系统而言，比较重要的有以下几点：

在前端/客户端设置秒杀答题，错开大量人下单的时间，阻止作弊器刷单
独立出来一套秒杀系统，专门负责处理秒杀请求
优先基于Redis进行高并发的库存扣减，一旦库存扣完则秒杀结束
秒杀结束之后，Nginx层过滤掉无效的请求，大幅度削减转发到后端的流量
瞬时生成的大量下单请求直接进入RocketMQ进行削峰，订单系统慢慢拉取消息完成下单操作

对于瞬时超高并发抢购商品的场景，**首先必须要避免直接基于数据库进行高并发的库存扣减**，因为那样会对库存数据库造成过大的压力

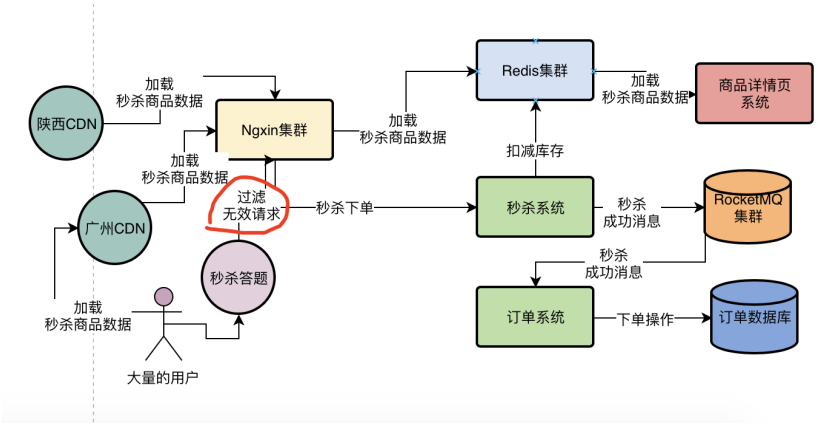
因为数据库单机可能每秒只能抗几千请求，但是改成直接基于Redis进行高并发扣减库存，每秒可以轻松抗几万请求。

我们看下面图的画圈的地方，这就是针对高并发的第一处优化，将瞬时高并发请求转发到Redis而不是MySQL，轻松抗下高并发。




一旦库存扣减为0之后，秒杀结束，因此实际上可能只有前面少量请求可以进入后台系统，**后续占据99%的请求，都可以直接在Nginx层面被拦截掉**，不会转发到后台系统造成任何压力

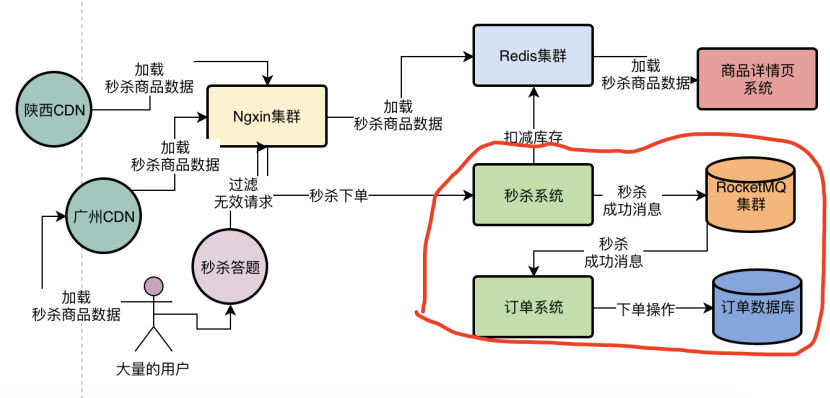
我们看下图中的画圈处。




接着瞬时生成的大量秒杀成功后的订单请求，不会直接交给订单系统去处理，否则也可能会对订单数据库瞬时造成过大压力

**此时会直接写入RocketMQ中进行削峰，让RocketMQ轻松抗下高并发压力，让订单系统慢慢消费和处理下单操作**

看下面图的画圈的地方。




所以通过上述分析，我们发现，像秒杀这种瞬时超高并发的场景，我们**架构优化的核心**就是独立出来一套系统专门处理，**避免高并发请求落在MySQL上**

因为MySQL天生不擅长抗高并发，我们需要通过Redis、Nginx、RocketMQ这些天生轻松可以单机抗几万甚至十万并发的系统来优化架构。

**8、一点小小的总结**

我们来对今天的文章做一点小小的总结

首先，我们利用两篇文章从一个比较高的角度给大家分析了秒杀场景下的各种问题，以及商品系统和订单系统需要进行如何的优化。

其次，大家要明白的一点是，我们在这个案例中，其实核心不是要给大家讲一个完整的秒杀架构的细节，而是让大家感受**在一个复杂系统架构中，RocketMQ是如何扮演削峰的角色的**

另外我们再次强调一下，一个秒杀系统的方方面面是很复杂的，我们不可能通过两篇文章把各种细节都讲清楚，至少需要一个完整专栏，用几十篇文章才能把一个秒杀系统落地的方方面面和细节都说清楚。

在我们的专栏里，核心还是利用这个案例给大家演示一下RocketMQ在高并发场景下削峰的使用，让大家明白RocketMQ在项目中是怎么来使用的。

## End



---

**狸猫技术窝其他精品专栏推荐：**

《从零开始带你成为JVM实战高手》

《21天Java 面试突击训练营》（分布式篇）（现更名为：**互联网Java工程师面试突击第2季**）

互联网Java工程师面试突击（第1季）

---

**重要说明：**

**如何提问**：每篇文章都有评论区，大家可以尽情在评论区留言提问，我会逐一答疑

**如何加群：**购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群？》**（购买后可见）**